videolinq

==========

API GUIDELINES FOR PROGRAMMERS

==========

Videolinq: Live API

The Videolinq Live Streaming API provides a programmer interface for creating Users, Stream Jobs, Endpoints, Schedulers, and Players programmatically. The API accepts JSON POST requests and returns responses as JSON.

The API is language agnostic and can be used with any programming language capable of sending HTTP POST requests and parsing JSON data. The following is a reference for using the API, and the operations available to application developers.

Videolinq accounts with API access share a "bucket" of available Stream Jobs, Endpoints, output hours, and a data transfer pool. Live video transcoding hours are billed separately. Accounts with API access can create an unlimited number of Sub-Users, Schedulers, and Players. IMPORTANT: When a Sub-User is assigned several Stream Jobs and Endpoints, this number is deducted from the master account total “bucket”. It is the responsibility of the programmer using the API feature on a Videolinq account to assign and limit the number of Stream Jobs, Endpoints, output hours and data transfer their Sub-Users can use.

The following features are managed by the Videolinq API:

1. User/Create new User and assign credentials.
2. User/Edit User.
3. User/Allow User to publish video and or closed captioning.
4. User/Remove User.
5. StreamJob/Crete new StreamJob.
6. StreamJob/Send a RTMP stream with unique stream key (stream id).
7. StreamJob/Use the StreamJob credentials to send closed captioning data.
8. StreamJob/Point to stream HDS/HLS paths.
9. StreamJob/Edit StreamJob.
10. StreamJob/Remove StreamJob.
11. Scheduler/Create new Scheduler.
12. Scheduler/Edit existing Scheduler.
13. Scheduler/Remove Scheduler.
14. Endpoint/Create new Endpoint.
15. Endpoint/Edit existing Endpoint.
16. Endpoint/Remove Endpoint.
17. Player/Create new player.

(Continue…)

18. Player/Edit existing player.
19. Player/Remove player.
20. Statistics/Output hours used per Account/User/Stream.
21. Statistics/Data transferred used per Account/User/Stream.
22. Statistics/Transcoder hours used per Account/User/Stream.

To use the API, you will need an active service account that will accept API calls, a unique key that identifies your account when API calls reach Videolinq, a username and password. Contact the sales department to request an API compatible service account.

Videolinq: Live Stream API Best Practices

This topic provides a guide to best practices for creating live streams using the Videolinq Live Stream API.

Overview

Videolinq provides a robust service for creating live streams. Streams created can be sent to multiple social media or streaming service providers. Streams can be played by the Videolinq Player, or 3rd party players supporting HTTP Live Streaming (HLS) and HTTP Dynamic Streaming (HDS). This guide outlines best practices for optimizing remote access and management of Videolinq services.

Videolinq Workflow

To use Videolinq customers must create a live stream in this order:

- Create a "User" with permissions to publish video, captions, or API access.
- Create "Endpoints". These are external target destination for the live stream. Endpoints are not required when using the Videolinq Player.
- Create a "Stream Job", assign a User that will be allowed to publish to the Stream Job, assign Endpoints as needed, and use a Scheduler as source for pulled streams. Stream Jobs can send video to both the Videolinq player, and to Endpoints.
- Create a "Player". The Videolinq Player can be used for live and video-on-demand streams. It can be used to stream video from a Videolinq Stream Job, or it can point to 3rd party streaming services. The Videolinq Player requires HDS and HLS paths to play across all browsers. It only needs a HTTP/HTTPS path to play video from Vimeo or YouTube.
- Create a "Scheduler" to pull RTMP/RTSP/HTTP source stream and assign it to a Stream Job. Schedules are activated manually, run 24/7, repeat at the same time every day, or can be assigned to run in a specific day/time.

- Add closed captioning to live streams: closed captioning can be added to live streams in two ways: closed captioning can be embedded on the SDI signal by a compatible hardware encoder, or closed captioning can be inserted to the live stream by one of Videolinq closed captioning partners. Adding closed captioning to a live stream is not supported by the Videolinq API.
- Insert advertising to live or archived video: Videolinq support VAST type of Ads insertion to live or archived streams via the Videolinq Player.

Input bandwidth

Providing a high-quality, stable input stream is the only way to ensure the best user experience for viewers. A good input stream provides the best video quality at the highest consistently available bandwidth from a location.

Minimum input bandwidth: 512 KBps
Maximum input bandwidth: 10 mbps

We recommend using the following bitrates for each screen resolution size:

240p – 512 KBps
360p – 1 MBps
720p – 1500-2500 KBps
1080i – 3-4 MBps
1080p – 4-5 MBps

Supported encoders

The Videolinq platform ingest Real Time Media Protocol (RTMP) video in a “push“ method. The platform can also “pull” a live video source by using the RTMP and RTSP paths. In addition, when using the “pull” method and pointing to a recorded file, a MPEG4 file encoded with the H.264 video and AAC audio codecs and pulled as RTMP/RTSP/HLS, can act as media source for live stream simulation.

Supported Platforms and CDNs

The Videolinq platform can publish live streams to many social media sites (example: YouTube, Facebook or Twitter), or to streaming video service providers (example: Akamai or Wowza Cloud).

Stream Connection - Retries

We recommend configuring the ingested streams to retry connecting on failure. Many retry attempts with a 5-10 second retry intervals will mitigate any intermittent connectivity issues between the encoder and the entry point. Extend the timeout settings accordingly. Set a maximum 20 attempt connection retry threshold, then stop trying. When the encoder is refused after the 20th attempt, display an error message to alert the User.

Concurrent Stream Jobs & Endpoints

The number of “Stream Jobs” and “Endpoints” is determined by the service plan ordered. When using the API features and operating query calls, the account will share "one bucket" with set amount of Stream Jobs, Endpoints, and data transfer allowance. All sub-Users created by the account via the API will share the limits allowed. An unlimited number of “Users”, “Schedulers”, and “Players” can be created with the API. Important note: when a Videolinq account is suspended or cancelled, all associated API records and media links will stop working.

Stream Job settings

Recommended input requirements for hardware or software encoders:

Protocol: RTMP
Video format: H.264
Audio format: AAC
Max audio sampling: 48000 Hz
Resolution: Up to 1080p
Bitrate: from 512 kbps to 10 mbps

Frame rate: 30 frames-per-second with keyframe interval every 30 frames or 1 second

Videolinq: Making Requests to the Live API

To use the Videolinq live API, send POST requests to the URL listed below. The JSON request body must be sent as a POST data.

Base Profile

All requests should be posted to the following URL:

https://my.videolinq.net/api

Authentication

For authentication and execution of actions, include with the request the Videolinq Account Service ID, the Videolinq User Name, and Password in the headers:

1. X-ACCOUNT-ID - Videolinq Account Service ID
2. X-ACCOUNT-USER-NAME - Videolinq User Name or Sub-User name
3. X-ACCOUNT-PASSWORD – User password or Sub-User password

The unique account information will be provided in the Welcome Email to the account owner. After creating Sub-Users, you will need to use Sub-Users credentials for creating records assigned to Sub-Users. Records created are determined by X-ACCOUNT-USER-NAME.

IMPORTANT: When a Videolinq account is suspended or cancelled, all associated API records and media links stop working.

General

The JSON request object must include the fields "operation" and "data". In the field "operation" provide the operation type. In the field "data" include the JSON object with data. This field can be empty for some requests:

```
{
   "operation":"Type of operation",
   "data":{ JSON object with data }
}
```

The following section describes the expected JSON request, and the JSON response that will be returned for each operation.

All responses will have an object "status" with a value of "success" or "fault". In the case of a successful operation, response will contain object named "data" with results of operation. The structure of the response data will vary depending on the request.

In the event of a failed request, the response will contain an object named "message" with a description of the cause for failure. Example of failure response:

```
{
   "message": "The reason for failure will be described here",
   "operation": "",
   "status": "fault"
}
```

To complete the API management tasks described in the next topics, you must have a master administrator credentials (provided when you open a service account with API access). To avoid confusion, we do not recommend performing mix operations using the Videolinq media dashboard and dynamic API actions.

IMPORTANT: When a Videolinq account is suspended or cancelled, all associated API records and media links stop working.

The operation descriptions described below illustrate responses assumed as success results.

Available API Operations

The following operations are supported by API requests to the Videolinq platform:

1. *User/Create new User and assign Password.*
2. *User/Enable or disable User.*
3. *User/Allow User to publish video and closed captioning.*
4. *User/Pull User unique ID.*
5. *User/Suspend User.*
6. *User/Remove User.*
7. *Endpoint/Create new Endpoint.*
8. *Endpoint/Update existing Endpoint.*
9. *Scheduler/Create or remove a Schedule.*
10. *Scheduler/Configure Schedule source.*
11. *Scheduler/Assign schedule method and activate.*
12. *StreamJob/Create a new StreamJob.*
13. *StreamJob/Update existing StreamJob.*
14. *StreamJob/Remove StreamJob.*
15. *Player/Create, edit, or remove a player.*
16. *Player/Configure player and assign video source.*
17. Statistics/Output hours used per Account/User/Stream.
18. Statistics/Data transferred used per Account/User/Stream.
19. Statistics/Transcoder hours used per Account/User/Stream.

Sub-Users created by the API will have with time lot of data associated with records they've created. Use the following parameters to query the information and return values for proper Sub-User management.

Action 1 – 6. Operations with account Sub-Users

These are the elements used in the sub-user object:

1. idPublic (String) - a unique user ID.
2. userName (String) - username for login.
3. password (String) - password for login.
4. displayName (String) - the username for displaying information.
5. jobLimit (Integer) - number of stream jobs allowed by the sub-user.
6. EndpointLimit (Integer) - number of endpoints allowed by sub-user.
7. isActive (Integer) - the sub-user status (0 - not active, 1 - active)
8. isEncoder (Integer) - sub-user can publish video (0 - not active, 1 - active)
9. isCaptionAgent (Integer) - sub-user can publish closed captioning separatly (0 - not active, 1 - active)*
10. isCaptionInVideo (Integer) - sub-user can publish closed captioning in video (0 - not active, 1 - active)**
11. isApiUser (Boolean) - system information about API sub-user status.
12. Id (Integer) - system user ID.

An example of a query made to request response for all fields:

```
{
 "idPublic":"",
 "userName":"",
 "password":"",
 "displayName":"",
 "jobLimit":10,
 "endpointLimit":10,
 "isActive":1,
 "isEncoder":1,
 "isCaptionAgent":1,
 "isCaptionInVideo":0,
 "isApiUser":true,
 "id":15
}
```


WHAT ARE SUB-USERS? Videolinq service accounts support two media management workflows: General account Users, and API based Users.

General accounts without API access include a Master Account Administrator that creates sub-users. These User profiles are assigned to video encoders or to a closed captioning service provider. Their permission level allows them to publish video or closed captioning to a specific StreamJob. General account Users can login to the media dashboard.

A Videolinq service account with API access has a Master Administrator login with the ability to see all the activity of sub-users created by the API. API Users have permission to create StreamJobs, Schedules, Endpoints and Players. The Master Administrator use the API to limit capacity of Sub-Users, to create usage reports, and to offer billable service levels based on the number of StreamJobs/Endpoints/data transfer allowed.

An example how to create a new Sub-User:

Send this request:

```
{
 "operation":"createUser",
 "data":{
   "displayName":"",
   "userName":"",
   "password":"",
   "jobLimit":0,
   "endpointLimit":0,
   "isActive":1,
   "isEncoder":1,
   "isCaptionAgent":1,
   "isCaptionInVideo":0
 }
}
```

The API will return this response:

```
{
 "data":{
   "isEncoder":1,
   "accountId":"",
   "password":"",
```



```
    "endpointLimit":0,
    "jobLimit":0,
    "displayName":"",
    "isApiUser":true,
    "idPublic":"",
    "id":-1,
    "userName":"",
    "isActive":1,
    "isCaptionAgent":1,
    "isCaptionInVideo":0
  },
  "message":"Account user created.",
  "operation":"createUser",
  "status":"success"
}
```


This is an example how to modify a Sub-User. The IdPublic value is used for the Sub-User identification:

Send this request:


```
{
  "operation":"modifyUser",
  "data":{
    "idPublic":"",
    "displayName":"",
    "userName":"",
    "password":"",
    "jobLimit":0,
    "endpointLimit":0,
    "isActive":1,
    "isEncoder":1,
    "isCaptionAgent":1,
    "isCaptionInVideo":0
  }
}
```


The API will return this response:


```
{
  "data":{
    "isEncoder":1,
    "accountId":"",
    "password":"",
    "endpointLimit":0,
    "jobLimit":0,
    "displayName":"",
    "isApiUser":true,
    "idPublic":"",
    "id":115,
    "userName":"",
    "isActive":1,
    "isCaptionAgent":1,
    "isCaptionInVideo":0
  },
  "operation":"modifyUser",
  "status":"success"
}
```


IMPORTANT: The "Delete Sub-User" action is irreversible.

An example of how to delete a Sub-User. The IdPublic value is used for identification. The field "data" in the object response will confirm the deleted Sub-User ID.

Send this request:

```
{
  "operation":"deleteUser",
  "data":{
    "idPublic":""
  }
}
```


The API will return this response:

```
{
 "data":"",
 "message":"Account user deleted.",
 "operation":"deleteUser",
 "status":"success"
}
```

An example of how to get a list of active sub-users for your account. In the response, the object field "data" will display a JSON array of sub-user objects.

Send this request:

```
{
 "operation":"getUserList",
 "data":{}       //warning: this object should be present
}
```

The API will return this response:

```
{
 "data":[
   {
     "isEncoder":1,
     "accountId":"",
     "password":"",
     "endpointLimit":4,
     "jobLimit":5,
     "displayName":"",
     "isApiUser":true,
     "idPublic":"",
     "id":15,
     "userName":"",
     "isActive":1,
     "isCaptionAgent":1,
     "isCaptionInVideo":0
   }
 ],
 "message":"User list retrieved successfully.",
 "operation":"getUserList",
 "status":"success"
}
```


WHAT ARE ENDPOINTS? Endpoints are created independently from StreamJobs, Users, Schedulers, and Players.

Endpoints are target records used by StreamJobs to send incoming video to target destinations. An Endpoint can only be used by one StreamJob at a time. Each Endpoint receives a unique identification number. Endpoint ID's are tracked to measure output billable time. When creating an Endpoint not listed in the destination table, use the RTMP or RTMP with authentication method to pass the required data fields.

Action 7 – 8. Create & Manage Endpoint Operations

These are the elements used in the Endpoint object:

1. idPublic (String) - unique endpoint id.
2. displayName (String) - endpoint name.
3. description (String) - endpoint description.
4. destination (Integer) - endpoint destination ID.*
5. hostName (String) - media server name or IP.
6. applicationName (String) - media server application name.
7. applicationInstanceName (String) - stream instance name.
8. streamName (String) - stream name or key assigned by streaming service.
9. userName (String) - stream username assigned for stream by streaming service.
10. password (String) - stream password assigned for stream by streaming service.
11. connectionQueryString (String) - values such as instance name or password that are part of the RTMP URL address.
12. streamQueryString (String) - values added to the stream name.
13. enableHttpOrigin (Integer) - make your stream available to Edgecast's HTTP CDN. (0 - not active, 1 - active)
14. streamId (String) - stream id assigned for stream by streaming service.
15. creatorName (String) - creator user display name.
16. creatorPublicId (String) - creator user IdPublic.
17. creationDate (String) - creation date.

* Identify independent Endpoint destinations by using different field numbers for objects:

Destination service name	Destination ID	Fields to use
Akamai HD	1	8,9,10,14
Verizon EdgeCast	2	5,8,10,13
Limelight	3	5,8,9,10
RTMP Server	6	5,6,7,8,11,12
RTMP Server with Authentication	7	5,6,7,8,9,10,11,12
Facebook Live	8	5,8
YouTube	9	5,8
Periscope	10	5,8
Livestream	11	5,8
Ustream	12	5,8
Twitch	13	5,8
Mixer	14	5,8
Dacast	15	5,8,9,10
Wowza Cloud	16	5,8,9,10

An example of a query field request:

```
{
  "hostName":"",
  "streamId":"",
  "displayName":"",
  "destination":0,
  "creatorName":"",
  "idPublic":"",
  "description":"",
  "userName":"",
  "streamName":"",
  "password":"",
  "enableHttpOrigin":0,
  "connectionQueryString":"",
  "creatorPublicId":"",
  "creationDate": "",
  "applicationInstanceName":"",
  "streamQueryString":"",
  "applicationName":""
}
```


Here is an example how to create an Endpoint:

Send this request:

```
{
 "operation":"createEndpoint",
 "data":{
   "displayName":"",
   "description":"",
   "userName":"",
   "password":"",
   "streamName":"",
   "streamId":"",
   "hostName":"",
   "applicationName":"",
   "applicationInstanceName":"",
   "connectionQueryString":"",
   "streamQueryString":"",
   "destination":5,
   "enableHttpOrigin":0
 }
}
```

The API will return this response:

```
{
 "data":{
   "hostName":"",
   "streamId":"",
   "displayName":"",
   "destination":5,
   "creatorName":"",
   "creationDate": "",
   "idPublic":"",
   "description":"",
   "userName":"",
   "streamName":"",
   "password":"",
   "enableHttpOrigin":0,
   "connectionQueryString":"",
   "applicationInstanceName":"",
```



```
    "streamQueryString":"",
    "applicationName":""
  },
  "message":"Endpoint created.",
  "operation":"createEndpoint",
  "status":"success"
}
```

An example how to modify an Endpoint. The object “IdPublic” is used for identification of the Endpoint.

Send this request:

```
{
  "operation":"modifyEndpoint",
  "data":{
    "idPublic":"",
    "displayName":"",
    "description":"",
    "userName":"",
    "password":"",
    "streamName":"",
    "streamId":"",
    "hostName":"",
    "applicationName":"",
    "applicationInstanceName":"",
    "connectionQueryString":"",
    "streamQueryString":"",
    "destination":1,
    "enableHttpOrigin":0
  }
}
```

The API will return this response:

```
{
  "data": {
    "hostName": "",
    "streamId": "",
    "displayName": "",
```



```
      "destination": 1,
      "creatorName": "",
      "idPublic": "",
      "description": "",
      "userName": "",
      "streamName": "",
      "accountId": "",
      "password": "",
      "enableHttpOrigin": 0,
      "connectionQueryString": "",
      "creatorPublicId": "",
      "creationDate": "",
      "applicationInstanceName": "",
      "streamQueryString": "",
      "applicationName": ""
    },
    "operation": "modifyEndpoint",
    "status": "success"
}
```


An example how to delete and Endpoint. The object “IdPublic” is used to identify the Endpoint. In the response object field "data" you will get the deleted endpoint IdPublic.

Send this request:


```
{
  "operation":"deleteEndpoint",
  "data":{
    "idPublic":""
  }
}
```


The API will return this response:


```
{
  "data": "",
  "message": "Endpoint deleted.",
  "operation": "deleteEndpoint",
  "status": "success"
}
```


An example how to get a list of Endpoints associated with the multiple Sub-Users. In the response, the object field "data" will return a JSON array of all Endpoints objects.

Send this request:

```
{
 "operation":"getEndpointList",
 "data":{}
}
```

The API will return this response:


```
{
  "data": [
    {
      "hostName": "",
      "streamId": "",
      "displayName": "",
      "destination": 6,
      "creatorName": "",
      "idPublic": "",
      "description": "",
      "userName": "",
      "streamName": "",
      "password": "",
      "enableHttpOrigin": 0,
      "connectionQueryString": "",
      "creatorPublicId": "",
      "creationDate": "",
      "applicationInstanceName": "",
      "streamQueryString": "",
      "applicationName": ""
    }
  ],
  "message": "Endpoint list retrieved successfully.",
  "operation": "getEndpointList",
  "status": "success"
}
```


HOW TO MANAGE SCHEDULES?

The Scheduler is used to automate scheduled live streams delivery to Endpoints. Scheduling pulls a video signal from H.264 encoders, IP Cameras, or MPEG-4 media files. When pulling video from a pre-recorded media file, we recommend that the file be encoded at high resolution using the H.264 video and AAC audio codecs.

Schedules have four methods of operations:

- They can be activated manually by the User.
- They can be set to run all day, all year, until stopped.
- They can be setup to repeat a start and stop a stream at specific day/time.
- They can be configured to start and stop a stream in various schedule every day of the week.

IMPORTANT: After a schedule is configured, the User must enable it in the media dashboard, and make it “Active”. A scheduler also can be started by using API command "startPullJob" (see stream job section). Based on the method selected in the Scheduler profile, the video will get pulled and begin playing.

Action 9 – 11. Create & Manage Schedules

These are the objects used in a Scheduler operation:

1. idPublic (String) - unique scheduler id.
2. displayName (String) - scheduler name.
3. Url (String) - scheduler RTMP or RTSP URL.
4. description (String) - scheduler description.
5. scheduleType (Integer) - scheduler type.
6. schedule (JSONArray) -array of arrays schedule objects for every day *
7. timeZone (String) - scheduler time zone (the time difference between UTC time and local time, in minutes. For example, if your time zone is GMT+2, time Zone will be -120). Note: The value is not a constant, because of the practice of using Daylight Saving Time.
8. creatorName (String) - creator user display name.
9. creatorPublicId (String) - creator user idPublic.
10. creationDate (String) - creation date.

Videolinq Scheduler Description	Scheduler ID Type
Manual. Never acquire automatically.	-1
24 Hours a day, 7 days a week.	0
Repeat the same schedule daily.	1
A different schedule for each day of the week.	2

* The schedule used for scheduler with scheduler type id 1 and 2

An explanation of the Schedule object:

1. day (Integer) - number of day (0 - Sunday, 1 - Monday, 2 - Tuesday, 3 - Wednesday, 4 - Thursday, 5 - Friday, 6 - Saturday). If the scheduler type is daily, it will use information from day 0 for all days.
2. startTime (Integer) – the scheduler start time in seconds from 12:00 AM. For example, if the time is 01:00 AM, the start time will be 3600.
3. endTime (Integer) – the scheduler end time in seconds from 12:00 AM.

Example of a schedule object:

```
{
  "day":0,
  "startTime":0,
  "endTime":300
}
```

Example of a Scheduler object:

```
{
   "id":0,
   "idPublic":"",
   "displayName":"",
   "url":"",
   "scheduleType":2,
   "schedule":[
     [
       {
         "day":0,
         "startTime":0,
         "endTime":300
```



```
        },
        {
          "day":0,
          "startTime":0,
          "endTime":60
        }
      ],
      [
        {
          "day":1,
          "startTime":0,
          "endTime":60
        }
       ],
      [ ],
      [ ],
      [ ],
      [ ],
      [ ]
    ],
    "description":"",
    "timeZone":"",
    "creatorName":"",
    "creatorPublicId":"",
    "creationDate": "",
    "accountId":""
  }
```

An example how to create a schedule:

Send this request:

```
{
  "operation":"createIpCameraSchedule",
  "data":{
    "displayName":"",
    "url":"",
    "scheduleType":0,
    "schedule":[
```



```
        [ ],
        [ ],
        [ ],
        [ ],
        [ ],
        [ ],
        [ ]
      ],
      "description":"",
      "timeZone":""
    }
}
```

The API will return this response:

```
{
  "data": {
    "accountId": "",
    "schedule": [
      [],
      [],
      [],
      [],
      [],
      [],
      []
    ],
    "scheduleType": 0,
    "displayName": "",
    "creatorPublicId": "",
    "creatorName": "",
    "creationDate": "",
    "idPublic": "",
    "description": "",
    "timeZone": "",
    "url": ""
  },
  "message": "Ip Camera Schedule created.",
  "operation": "createIpCameraSchedule",
  "status": "success"
}
```


An example how to modify a Scheduler entry. The object "IdPublic" is used to identify the scheduler.

Send this request:

```
{
 "operation":"modifyIpCameraSchedule",
 "data":{
   "idPublic":"",
   "displayName":"",
   "url":"",
   "scheduleType":0,
   "schedule":[
     [ ],
     [ ],
     [ ],
     [ ],
     [ ],
     [ ],
     [ ]
   ],
   "description":"",
   "timeZone":""
 }
}
```

The API will return this response:

```
{
  "data": {
    "accountId": "",
    "schedule": [
      [],
      [],
      [],
      [],
      [],
      [],
      []
    ],
    "scheduleType": 0,
    "displayName": "",
    "creatorName": "",
```



```
      "creationDate": "",
      "idPublic": "",
      "description": "",
      "timeZone": "",
      "url": ""
    },
    "message": "Ip Camera Schedule modified.",
    "operation": "modifyIpCameraSchedule",
    "status": "success"
}
```


An example how to delete a Scheduler record. The object "IdPublic" is used to identify the scheduler. In the response object field "data", look for the deleted scheduler IdPublic.

Send this request:


```
{
  "operation":"deleteIpCameraSchedule",
  "data":{
    "idPublic":""
  }
}
```


The API will return this response:


```
{
  "data": "",
  "message": "Ip Camera Schedule deleted.",
  "operation": "deleteIpCameraSchedule",
  "status": "success"
}
```


An example how to get a list of schedulers for the entire account. Look for the response object "data" field, to see a JSON array of all schedulers objects.

Send this request:

```
{
  "operation":"getIpCameraScheduleList",
  "data":{}
}
```

The API will return this response:

```
{
"data": [
    {
      "schedule": [
        [],
        [],
        [],
        [],
        [],
        [],
        []
      ],
      "scheduleType": 2,
      "displayName": "",
      "creatorPublicId": "",
      "creatorName": "",
      "creationDate": "",
      "idPublic": "",
      "description": "",
      "url": "",
      "status": "on"
    },
],
  "message": "IP Camera Schedule list retrieved successfully",
  "operation": "getIpCameraScheduleList",
  "status": "success"
}
```


WHAT ARE STREAM JOBS?

Stream Jobs are publishing channels. A Stream job requires at least one User assigned. Stream Jobs act as a source for media paths used to create players, or act as a distribution hub that publish live streams Endpoint destinations.

Action 12 – 14. Create & Manage StreamJobs

These are the objects used in the creation of a Stream Job:

1. idPublic (String) - unique stream job id.
2. displayName (String) - stream job name.
3. Description (String) - stream job description.
4. sourceType (Integer) - stream job source type (0 - RTMP push option, 1 - pull scheduler).
5. ipCameraScheduleId (String) - Scheduler idPublic (used if source type 1).
6. noAudio (Boolean) – when audio is not present, a silent track is added to the stream.
7. endpointIds (JSON array) – list array of Endpoints idPublic.
8. userIds (JSON array) – list array of sub-users idPublic.
9. Transcoder (JSON array) – list array of transcoder objects *
10. creatorName (String) - display user name.
11. creatorPublicId (String) – display the user idPublic.
12. creationDate (String) - creation date.
13. isStarted (Boolean) – show StreamJob status.

StreamJobs support video ingestion in two ways:

Push Method

The stream is sent from a video source such as camera or video encoder using the RTMP protocol. When the incoming video is detected, the job and associated settings start automatically. An input RTMP stream looks like this -

RTMP path: *rtmp://in.videolinq.net/ingest?user={username}@{Account id}:{password}*
Stream name/key: *{Stream job idPublic}@{Account id}*

Closed captioning (subtitles) insertion method are dictated in two ways

1. Subtitles are sent with the video (isCaptionInVideo user field):
   For this type of closed captions video need to be sent to
   rtmp://in.videolinq.net/ingest-2?user={username}@{Account id}:{password}

2. Subtitles are sent separately (isCaptionAgent user field):
   Closed captions will be sent by 3rd party partner programs to:
   Host: *http://in.videolinq.net/caption*
   Stream Job: *{Stream job idPublic}@{Account id}*
   User: *{username}@{Account id}:{password}*

Pull Method

The stream is pulled by the Scheduler. The schedule points to a RTMP/RTSP/HLS media source. The StreamJob will be activated based on one of the four schedule settings.

Video Paths

After a Stream Job is created the live stream can be pulled from Videolinq by using:
HDS URL: *https://out.videolinq.net/stream/{Stream job idPublic}@{Account id}/manifest.f4m*
HLS URL: *https://out.videolinq.net/stream/{Stream job idPublic}@{Account id}/playlist.m3u8*

TRANSCODING OPTION?

Transcoding allow the instant conversion of video resolution and bitrate to another resolution and bitrate. Transcoder settings are created in the StreamJob and assigned to specific Endpoints. Transcoding jobs are billed per Stream Job associated with a master Videolinq account. Additional fee per hour applies.

Sample of transcoder objects used in StreamJobs:

1. displayName (String) - transcoder name.
2. vcodecprofile (Integer) - H.264 Profile (0 - No video, 1 - Baseline, 2 - Main)
3. vbitrate (Integer) - video bitrate (in kbps).
4. vframerate (Integer) - video FPS (0 - Full, 1 - Half).
5. abitrate (Integer) - audio bitrate (in kbps). Used if vcodecprofile - no video.
6. width (Integer) - video width.
7. height (Integer) - video height.
8. endpointIds (JSONArray) - array of endpoints idPublic.

An example of a Transcoder object:

```
{
  "displayName":"",
  "vcodecprofile":0,
  "vbitrate":0,
  "vframerate":0,
  "abitrate":256,
  "width":720,
  "height":405,
  "endpointIds" [ ]
}
```

An example of a StreamJob object:

```
{
  "idPublic":"",
  "displayName":"",
  "description":"",
  "sourceType":1,
  "userIds":[
    ""
  ],
  "endpointIds":[
    ""
  ],
  "ipCameraScheduleId":"",
  "creatorName":"",
  "creatorPublicId":"",
  "creationDate": "",
  "noAudio":false,
  "transcoder":[

 {
      "displayName":"",
      "vcodecprofile":2,
      "vbitrate":768,
      "vframerate":1,
      "abitrate":0,
      "width":720,
      "height":405,
```



```
        "endpointIds":[
          ""
        ]
      }
    ],
    "endpoints":[

    ],
    "isStarted":false,
    "users":[

    ],
    "accountId":""
}
```

An example how to create a new StreamJob.

Send this request:

```
{
  "operation":"createJob",
  "data":{
    "displayName":"",
    "description":"",
    "sourceType":"",
    "userIds":[
      ""
    ],
    "endpointIds":[
      ""
    ],
    "ipCameraScheduleId":"",
    "noAudio":true,
    "transcoder":[]
  }
}
```


The API will return this response:


```
{
  "data": {
    "displayName": "",
    "ipCameraScheduleId": "",
    "creatorName": "",
    "idPublic": "",
    "description": "",
    "isStarted": false,
    "endpointIds": [
      ""
    ],
    "accountId": "",
    "sourceType": 1,
    "creatorPublicId": "",
    "creationDate": "",
    "noAudio": true,
    "userIds": [
      ""
    ],
    "transcoder": []
  },
  "message": "Stream job created.",
  "operation": "createJob",
  "status": "success"
}
```


An example how to modify a StreamJob. The object "IdPublic' is used to identify the StreamJob.

Send this request:


```
{
 "operation":"modifyJob",
 "data":{
   "idPublic":"",
   "displayName":"",
   "description":"",
   "sourceType":1,
   "userIds":[
     ""
```



```
    ],
    "endpointIds":[
      ""
    ],
    "ipCameraScheduleId":"",
    "creatorName":"",
    "creatorPublicId":"",
    "noAudio":true,
    "transcoder":[],
    "isStarted":false
  }
}
```

The API will return this response:

```
{
  "data": {
    "displayName": "",
    "ipCameraScheduleId": "",
    "creatorName": "",
    "idPublic": "",
    "description": " ",
    "isStarted": false,
    "endpointIds": [
      ""
    ],
     "sourceType": 1,
    "creatorPublicId": "",
    "creationDate": "",
    "noAudio": true,
    "userIds": [
      ""
    ],
    "transcoder": []
  },
  "message": "Stream job modified.",
  "operation": "modifyJob",
  "status": "success"
}
```


An example how to delete a StreamJob. The object "IdPublic" is used for identifying the StreamJob. The response "data" field object provides the deleted StreamJob IdPublic.

Send this request:

```
{
 "operation":"deleteJob",
 "data":{
   "idPublic":""
 }
}
```

The API will return this response:

```
{
  "data": "",
  "message": "Stream Job deleted.",
  "operation": "deleteJob",
  "status": "success"
}
```

An example how to get a list of StreamJobs for your account. The "data" field object response will provide a JSON array of all StreamJob objects.

Send this request:

```
{
 "operation":"getJobList",
 "data":{}
}
```

The API will return this response:

```
{
  "data": [
    {
      "displayName": "",
      "ipCameraScheduleId": "",
      "creatorName": "",
      "idPublic": "",
```



```
            "description": "",
            "isStarted": false,
            "endpointIds": [
               ""
            ],
            "accountId": "",
            "sourceType": 1,
            "creatorPublicId": "",
            "creationDate": "",
            "noAudio": false,
            "userIds": [
               ""
            ],
            "transcoder": []
         }
      ],
      "message": "Stream Job list retrieved successfully.",
      "operation": "getJobList",
      "status": "success"
   }
```


Example of how to get a list of active jobs. The "data" field object response provides a JSON array list of all active StreamJobs.

Send this request:


```
   {
      "operation":"getActiveJobs",
      "data":{}
   }
```


The API will return this response:


```
   {
      "code": 200,
      "data": [
         {
            "displayName": "",
            "ipCameraScheduleId": "",
```



```
            "creatorName": "",
            "idPublic": "",
            "description": "",
            "isStarted": ,
            "creationDate": "",
            "endpointIds": [
              "",
              ""
            ],
            "accountId": "",
            "sourceType": 1,
            "creatorPublicId": "",
            "noAudio": ,
            "userIds": [
              ""
            ],
            "transcoder": []
        }
    ],
    "message": "Request processed.",
    "operation": "getActiveJobs",
    "status": "success"
}
```


An example how to start a StreamJob, using the pull method:

1. IdPublic – idPublic stream job for pulling.
2. timeZone (String) - your offset (the time difference between UTC time and local time, in minutes. For example, if your time zone is GMT+2, time Zone will be -120). This is important for StreamJobs with a daily and weekly scheduler.

Send this request:

```
{
  "operation":"startPullJob",
  "data":{
    "idPublic":"",
    "timeZone":""
  }
}
```


The API will return this response:

```
{
   "code": 200,
   "message": "Request processed.",
   "operation": "startPullJob",
   "status": "success"
}
```

An example of how to stop a StreamJob using the pull method.

Send this request:

```
{
  "operation":"stopPullJob",
  "data":{
    "idPublic":""
  }
}
```

The API will return this response:

```
{
   "code": 200,
   "message": "Request processed.",
   "operation": "stopPullJob",
   "status": "success"
}
```

HTML5 PLAYER WIZARD

API Sub-Users can create unlimited number of players pointing to live streams services by Videolinq or 3rd party streaming providers, to recorded files hosted by any media server, and to Vimeo or YouTube videos. When a HTML5 player is created, generate an iFrame link. The iFrame link can be embed in any website.

Action 17 – 19. Create & Manage Players

These are the objects used in the creation of the video player:

1. name (String) - player name.
2. id (Integer) - unique player id.
3. idPublic (String) - unique player idPublic.
4. description (String) - player description.
5. creatorPublicId (String) - creator user idPublic.
6. creatorName (String) - creator user display name.
7. isLive (Integer) - player video type (0 - Video-on-Demand, 1 - Live Stream)
8. sourceType (Integer) - player input source type (0 - HDS/HLS path, 1 - Vimeo/YouTube path).
9. hdsPath (String) - player input hdsPath (0 sourceType).
10. hlsPath (String) - player input hlsPath (0 sourceType).
11. vimeoYoutubePath (String) - player input vimeoYoutubePath (1 sourceType).
12. edgyIcons (Integer) - display edgy icons (0 - No, 1 - Yes).
13. outlinedIcons (Integer) - display outlined icons (0 - No, 1 - Yes).
14. disableVolume (Integer) - hide volume control (0 - No, 1 - Yes).
15. enableMute (Integer) - Enable mute button (0 - No, 1 - Yes).
16. isMuted (Integer) - start video with audio mute (0 - No, 1 - Yes).
17. fixedControls (Integer) - fixed control elements (0 - No, 1 - Yes).
18. enableCaptions (Integer) - enable captions (0 - No, 1 - Yes).
19. captionsLeft (Integer) - move captions to top left position (default bottom center position).(0 - No , 1 - Yes).
20. enableSettings (Integer) - enable settings button (0 - No, 1 - Yes). (No - quality button).
21. enableFullscreen (Integer) - enable full screen (0 - No , 1 - Yes).
22. share (Integer) - enable share button (0 - No , 1 - Yes).
23. logo (String) - custom logo picture link.
24. logoOpacity (Integer) - custom logo opacity.
25. controlsBg (String) - control bar background (RGBA color space. Example: rgba (255, 255, 255, 0.6)
26. timelineBg (String) - Control bar VOD timeline (RGBA color space).
27. progressBg (String) - progress bar background (RGBA color space).
28. bufferBg (String) - Control bar buffer (RGBA color space).
29. subtitleBg (String) subtitle background (RGBA color space).
30. subtitleColor (String) - subtitle color (RGBA color space).
31. containerStyle (String) - field used for saving generated code by system.
32. cssStyle (String) - field used for saving generated code by system.
33. skinIndex (Integer) - skin type (0 - Normal, 1 - Minimal, 2 - Playful)

34. presetIndex - skin preset type (0 - Hover Controls, Default Timeline 1 - Fixed Controls, Default Timeline 2 - Hover Controls, Slim Timeline 3 - Fixed Controls, Slim Timeline 4 - Hover Controls, Full Timeline 5 - Fixed Controls, Full Timeline 6 - Hover Controls, Fat Timeline 7 - Fixed Controls, Fat Timeline)
35. iframeHeight (String) - frame height *
36. iframeWidth (String) - frame width *
37. creationDate (String) - creation date.
38. enableAutoplay (Integer) - autoplay (0 - No, 1 - Yes)
39. poster (String) - poster before video link.
40. aspectRatio (Integer) - aspect ratio (0 - 16:9, 1 - 4:3)
41. enableAdvertising (Integer) - enable advertising (0 - No, 1 - Yes).
42. advertSourceType (Integer) - advert source type (0 - URL path to the campaign tag, 1 - XML data with additional configuration).
43. advertPlacementType (Integer) - advert placement type (0 - pre-roll, 1 - mid-roll, 2 - post-roll) (using when advertSourceType - 0).
44. advertTime (Integer) - Mid-Roll time (in seconds) (using when advertSourceType - 0 and advertPlacementType - 1).
45. advertUrl (String) - URL advert. (using when advertSourceType - 0)
46. advertXml (String) - XML advert. (using when advertSourceType - 1)

* These fields must be inserted in the player insert code.

An example of a Player insert code generation:

<iframe width={iframeWidth} height={iframeHeight} scrolling='no' frameborder='0' allowfullscreen src='https://my.videolinq.net/player?idPublic={player idPublic}></iframe>

Example player object:

```
{
    "fixedControls": 1,
    "advertTime": 0,
    "enableSettings": 1,
    "outlinedIcons": 1,
    "idPublic": "",
    "description": "",
    "aspectRatio": 0,
    "enableAutoplay": 0,
```



```
    "advertPlacementType": 0,
    "edgyIcons": 1,
    "enableFullscreen": 1,
    "disableVolume": 1,
    "isLive": 0,
    "enableAdvertising": 0,
    "share": 1,
    "presetIndex": 4,
    "iframeHeight": "",
    "id": 406,
    "hdsPath": "",
    "advertSourceType": 0,
    "bufferBg": "",
    "skinIndex": 2,
    "advertUrl": "",
    "controlsBg": "",
    "containerStyle": "",
    "cssStyle": "",
    "enableMute": 1,
    "creationDate": "",
    "captionsLeft": 1,
    "vimeoYoutubePath": "",
    "accountId": "",
    "subtitleBg": "",
    "enableCaptions": 1,
    "timelineBg": "",
    "subtitleColor": "",
    "sourceType": 0,
    "iframeWidth": "",
    "creatorPublicId": "",
    "name": "",
    "hlsPath": "",
    "advertXml": "",
    "poster": "",
    "isMuted": 0,
    "progressBg": ""
}
```


An example how to create a dynamic player code:

Send this request:


```
{
  "operation":"createPlayer",
  "data":{
    "name":"",
    "description":"",
    "sourceType":"0",
    "hdsPath":"",
    "hlsPath":"",
    "vimeoYoutubePath":"",
    "edgyIcons":1,
    "outlinedIcons":1,
    "disableVolume":1,
    "enableMute":1,
    "isMuted":0,
    "fixedControls":1,
    "enableCaptions":1,
    "captionsLeft":1,
    "isLive":0,
    "enableSettings":1,
    "share":1,
    "key":"",
    "logo":"",
    "logoOpacity":0,
    "controlsBg":"",
    "timelineBg":"",
    "progressBg":"",
    "bufferBg":"",
    "subtitleBg":"",
    "subtitleColor":"",
    "containerStyle":"",
    "cssStyle":"",
    "skinIndex":2,
    "presetIndex":4,
    "iframeHeight":"",
    "iframeWidth":"",
    "creationDate":"",
    "enableAutoplay":0,
    "poster":"",
```



```
    "aspectRatio":0,
    "enableAdvertising":0,
    "advertPlacementType":0,
    "advertTime":0,
    "advertSourceType":0,
    "advertUrl":"",
    "advertXml":"",
    "enableFullscreen":1,
  }
}
```

The API will return this response:

```
{
   "data": {
      /* created player object */
             },
   "message": "Player wizard created.",
   "operation": "createPlayer",
   "status": "success"
}
```

An example how to modify a player. The object "id" is used to identify the player:

Send this request:

```
{
  "operation":"modifyPlayer",
  "data":{
    "id":1,
    "name":"",
    "description":"",
    "sourceType":"0",
    "hdsPath":"",
    "hlsPath":"",
    "vimeoYoutubePath":"",
    "edgyIcons":1,
    "outlinedIcons":1,
```



```
      "disableVolume":1,
      "enableMute":1,
      "isMuted":0,
      "fixedControls":1,
      "enableCaptions":1,
      "captionsLeft":1,
      "isLive":0,
      "enableSettings":1,
      "share":1,
      "key":"",
      "logo":"",
      "logoOpacity":0,
      "controlsBg":"",
      "timelineBg":"",
      "progressBg":"",
      "bufferBg":"",
      "subtitleBg":"",
      "subtitleColor":"",
      "containerStyle":"",
      "cssStyle":"",
      "skinIndex":2,
      "presetIndex":4,
      "iframeHeight":"",
      "iframeWidth":"",
      "creationDate":"",
      "enableAutoplay":0,
      "poster":"",
      "aspectRatio":0,
      "enableAdvertising":0,
      "advertPlacementType":0,
      "advertTime":0,
      "advertSourceType":0,
      "advertUrl":"",
      "advertXml":"",
      "enableFullscreen":1,
      "creatorName":"",
      "creatorPublicId":""
    }
  }
```


The API will return this response:

```
{
   "data": {
     /* modified player object */
              },
   "message": "Player wizard created.",
   "operation": "createPlayer",
   "status": "success"
```

An example how to delete a player. The object “id” is used to identify the player. The “data” object field response will provide the deleted player “id”.

Send this request:

```
{
  "operation":"deletePlayer",
  "data":{
     "id":1
  }
}
```

The API will return this response:

```
{
   "data": 1,




   "message": "Player wizard deleted.",
   "operation": "deletePlayer",
   "status": "success"
}
```


An example how to get the list of all players. The "data" object field response will provide a JSON array listing all players.

Send this request:

```
{
  "operation":"getPlayerList",
  "data":{ }
}
```

The API will return this response:

```
{
   "data": [
          /* array of player objects */
   ],
   "message": "player list retrieved successfully.",
   "operation": "getPlayerList",
   "status": "success"
}
```

Action 20 – 22. Pull & Display Analytics

Videolinq service accounts with API access use centralized service and data transfer "buckets". The total number of StreamJobs, Endpoints, and data transfer is shared among all Sub-Users created by the account via the API. It is recommended to provide each Sub-User with their specific number of allowed Stream Jobs, Endpoints, and data transfer. Live video transcoding hours are billed only when a Sub-User account activate transcoding inside a Stream Job. Using the following actions, you will be able to pull and display usage activity for a specific stream-name of a Sub-User, and their monthly Endpoint output video hours, transcoding jobs, and data transfer used by Players they create.

Here is an example how to get the account StreamJob limits:

a. maxInboundStreams (Integer) - maximum count of stream jobs.
b. usedInboundStreams (Integer) - count of used stream jobs. (include stream jobs assigned to sub users)

c. maxOutboundStreams (Integer) - maximum count of endpoints. (include stream jobs assigned to sub-users)
d. usedOutboundStreams (Integer) - count of used Endpoints.

Send this request:

```
{
  "operation":"getStreamLimits",
  "data": {}
}
```

The API will return this response:

```
{
   "data": {
      "usedOutboundStreams": ,
      "usedInboundStreams": ,
      "maxInboundStreams": ,
      "maxOutboundStreams":
   },
   "isApiUser": false,
   "isApiAccount": true,
   "message": "Streams info reloaded",
   "operation": "getStreamLimits",
   "status": "success"
}
```

An example how to get output hours and data for the current month:

e. billingHours (Integer) - billing hours current month.
f. billingData (Integer) - billing data current month in GB.

Send this request:

```
{
  "operation":"getBillingInfo",
  "data":{}
}
```


The API will return this response:


```
{
   "operation": "getBillingInfo",
   "billingHours": 0,
   "billingData": 0,
   "status": "success"
}
```


ABOUT THE VIDEOLINQ LIVE STREAM API – LEGAL DISCLAIMER

Disclaimer of Warranty. Unless required by applicable law or agreed to in writing, Videolinq provides this Application Programming Interface (API) on an "AS IS" BASIS, WITHOUT WARRANTIES OR CONDITIONS OF ANY KIND, either express or implied, including, without limitation, any warranties or conditions of TITLE, NON-INFRINGEMENT, MERCHANTABILITY, or FITNESS FOR A PARTICULAR PURPOSE. You are solely responsible for determining the appropriateness of using or redistributing the API and assume any risks associated with Your exercise of permissions under this License. For additional information please refer to our Terms of Service as listed at https://videolinq.com/terms.

